ON THE IMPORTANCE OF EXACT SYNCHRONIZATION FOR DISTRIBUTED AUDIO SIGNAL PROCESSING

Rainer Lienhart ⋆*, Igor Kozintsev* ⋆*, Stefan Wehr* ⋆⋆*, Minerva Yeung* ⋆

⋆ Intel Corporation
Microprocessor Research, Intel Labs
3600 Juliette Lane, Santa Clara, CA 95052, USA
{ Rainer.Lienhart, Igor.V.Kozintsev, Minerva.Yeung }@intel.com

⋆⋆ University Erlangen-Nürnberg
Multimedia Communications and Signal Processing
Cauerstraße 7, 91058 Erlangen, Germany
Wehr@lnt.de

ABSTRACT

We propose a new paradigm for implementations of audio array processing algorithms on a network of distributed general-purpose computers. In contrast to currently existing DSP processor-based solutions, our approach offers new possibilities for advanced array signal processing by enabling the usage of general-purpose computing platforms with their superior computational and storage resources. We demonstrate that synchronization of sensors is essential for acoustic Blind Source Separation (BSS) algorithms, and we propose a synchronization scheme that enables BSS on distributed, wirelessly networked computers and can easily be implemented on existing hardware.

1. INTRODUCTION AND MOTIVATION

During the last years, significant progress was made in audio array processing [1, 2]. It enabled affordable consumer devices for hands-free acoustic human-machine interfaces for, e.g., speech recognition, video/audio conferencing, voice over IP, e.g [3]. The existing products are currently dominated by dedicated ASIC/DSP hardware. The rationale behind DSP implementations is obvious: they guarantee short latency, precise synchronization, dedicated computing resources and interconnections between components. However, there are also disadvantages of DSP-based implementations such as their fixed functionality, considerable cost, geometrical constraints (on the placement of sensors/actuators) and limited software development/debugging capabilities.

In this paper we demonstrate that audio array algorithms can be implemented using an alternative architecture such as distributed network of general-purpose computers (GPC) with onboard sensors. Examples of such platforms are laptops, PDAs, tablets, etc., with integrated audio and wireless networking capabilities. In contrast to DSP-based systems, GPCs are multifunctional, scalable and capable of performing more complex computational tasks. However, before these devices can be used for audio array processing several problems need to be resolved. For example, the lack of precise synchronization between wirelessly connected computers is a serious issue for audio processing algorithms where sometimes (as we show in this paper) the sample time accuracy has to have a microsecond precision. Besides synchronization other obstacles for using GPCs as sensor/actuator nodes are variable latency of data processing, non-guaranteed link bandwidth, and the need to share computing/communication resources with other applications. It is currently quite challenging to implement existing array processing algorithms on GPCs, and we believe that the solution in general would require significant modifications of both the GPCs and the array processing algorithms. However in this paper, we show an example of an implementation of array audio algorithms on existing distributed GPCs.

This work demonstrates how distributed GPCs can be applied to array signal processing using as example BSS. BSS has already been proven feasible on a single computing platform (see e.g., [4]). To the best of our knowledge BSS (and, in fact, other array audio algorithms) has never been implemented in a distributed computing environment as we propose in this paper. We demonstrate how BSS can be implemented on real distributed computing platforms and how the problem of sensor synchronization may be addressed.

2. PROBLEM AND PROPOSED APPROACH

2.1. BSS applications

BSS is an approach to solve the so-called cocktail party problem: N speakers are recorded by $M \geq N$ microphones (Fig. 3). Each microphone signal is a mixture of non-stationary and reverberated signals. Furthermore, speaker and microphone positions are generally unknown.

Several applications require to separate source signals from the mixture. Examples include hearing aids, speech recognition, and teleconference systems where we may want to focus on a particular speaker only. Current BSS applications implicitly assume recording by a single multichan-

nel audio device (with synchronized sampling in all channels). Our goal, however, is to use the built-in microphones of several distributed GPCs to record audio signals. Such distributed setup leads to a severe problem: The sampling rates of different audio devices do not exactly match each other anymore. This leads to significant degradation of the performance [1]. Figure 1 shows the effect of sampling rate differences in a two-speaker/two-microphone setting on the performance of the BSS. Even a 1 Hz sampling frequency offset results in a 5 dB performance loss.

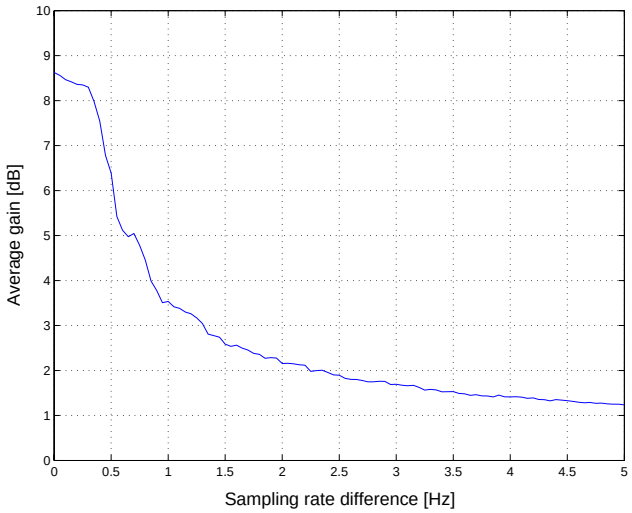

Fig. 1. Sensitivity of BSS to sampling rate differences. Even a 1 Hz sampling frequency offset results in a 5 dB performance loss.

Due to this high algorithm sensitivity to sampling rate deviations, synchronization is essential for distributed signal processing. We solve this problem by distributing common clock information to all devices as described in 2.2. Every laptop (or PDA) recovers the sync signal and realigns audio samples from different sensors. Synchronized signals are then processed in the computer (outside or inside the "audio-recording-network") to perform BSS.

2.2. BSS on distributed platforms

The BSS algorithm is highly sensitive to small differences in sample frequency as demonstrated in Figure 1. Differences in sample frequency, however, are unavoidable on distributed platforms and must therefore be (locally) corrected before applying BSS.

In our synchronization solution we assume that audio channels on the same I/O device (e.g., PCMCIA card, internal multichannel ADC) are synchronized and in abundance available on future computing platforms (e.g., 4 to 8 audio inputs). The main idea of our novel synchronization scheme is to 'misuse' a single audio channel for distributing global synchronization information. Synchronization signals are formed in a master unit using its own clock to modulate an audio carrier signal. The carrier signal can be chosen from many possible types. We use Maximum Length Sequences (MLS, [6]) because of their good autocorrelation characteristics. The synchronization signals are delivered to the distributed computing platform using dedicated links with small stable latency [2] such as wireless RF channels. In our case, a simple FM radio transmitter and multiple receivers are used to analog modulate/demodulate the audio sync signals.

Using an RF channel for distribution of the audio sync signals rather than air is a critical component of our synchronization scheme. If the sync signals are sent through the air, synchronization cannot be achieved due to the significant difference in propagation time. Additionally, synchronization signals will interfere with the audio scene and degrade the recording quality. Our solution, however, requires an additional audio input channel dedicated solely to the audio sync (timestamp) signal. Since the sync signals are processed in dedicated circuits and delivered by electromagnetic waves, propagation time is small and almost constant. Figure 2 illustrates our setup. One GPC distribute the MLS

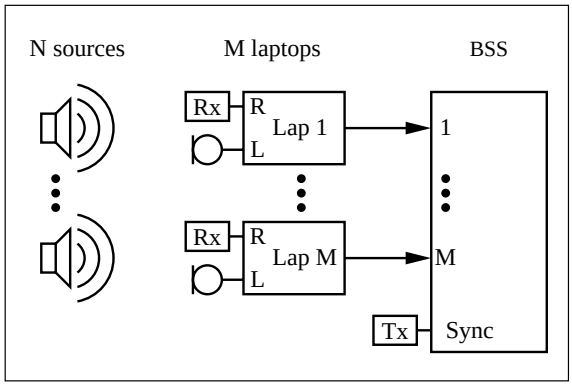

Fig. 2. Setup of distributed BSS platform

sync signal with modulated sequence numbers via the RF channel (Tx). These MLS sequences are received at each distributed GPC, demodulated to audio, and are fed into dedicated audio channel, while the respective microphone signal (mono) is captured on another audio channel. After processing the sync audio track (we use a simple correlation receiver to determine the start of each MLS), we can determine the locations of timestamps and perform sampling rate conversion. Fortunately, all operations can be performed locally, and therefore our synchronization scheme scales well

[1] In this work we use our implementations of BSS presented in [4, 5].

[2] Note that we only need a relative synchronization of all the inputs. Low latency is desired but not always required.

with the number of computing platforms and microphones. Finally, preprocessed audio tracks are delivered to the computing platform to perform BSS.

3. EXPERIMENTAL RESULTS

For our experiments, we implemented two BSS approaches, published by Lucas Parra and others [4, 5], as real-time systems. Two different cost functions are incorporated in [4] and [5], which are denoted as J_1 and J_2 in this work.
Figure 3 gives an overview of BSS. $\mathbf{A}(k)$ captures room impulse responses $a_{ji}(k)$ between source signal $s_i(k)$ and microphone signal $x_j(k)$. $\mathbf{W}(k)$ consists of filters $w_{lj}(k)$ between microphone signal $x_j(k)$ and estimated source signal $\hat{s}_i(l)$. In contrast to [4] and [5], we compute the signal

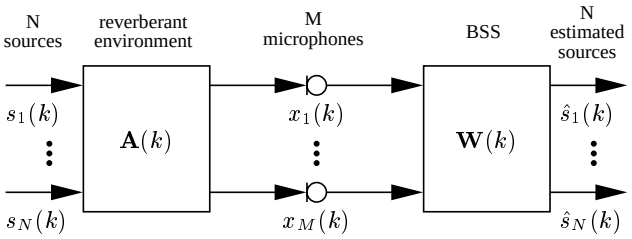

Fig. 3. Block diagram of the BSS problem with mixing matrix $\mathbf{A}(k)$ and unmixing matrix $\mathbf{W}(k)$.

to interference ratio (SIR) as follows:

$$\mathrm{SIR}_{in,i}(k) = 10\log_{10}\left[\frac{\mathrm{cov}\left\{y_{s_i,m_i}(k)\right\}}{\mathrm{cov}\left\{\sum_{l\neq i} y_{s_l,m_i}(k)\right\}}\right] \quad (1)$$

$$\mathrm{SIR}_{out,i}(k) = 10\log_{10}\left[\frac{\mathrm{cov}\left\{\sum_{j} y_{s_i,m_j,\hat{s}_i}(k)\right\}}{\mathrm{cov}\left\{\sum_{l\neq i}\sum_{j} y_{s_l,m_j,\hat{s}_i}(k)\right\}}\right] \quad (2)$$

In both equations, the numerator consists of the desired signal $s_i(k)$ and the denominator of all interfering signals $s_l(k)$ ($l \neq i$). $y_{s_i,m_j}(k) = s_i(k) * a_{ji}(k)$ represents source i recorded by microphone j. Accordingly, $y_{s_i,m_j,\hat{s}_l}(k) = s_i(k) * a_{ji}(k) * w_{lj}(k)$ represents the contribution of source i, recorded by microphone j, to estimated source l. Comparing the input and output SIR, we obtain the gain due to BSS in dB:

$$\mathrm{Gain}_i(k) = \mathrm{SIR}_{out,i}(k) - \mathrm{SIR}_{in,i}(k) \quad (3)$$

A top view of the simulation setup is depicted in figure 4. The microphones and loudspeakers were set up in a sound-control chamber (with short room impulse response and low noise level). In order to minimize background noise, we placed playback and recording devices outside of the chamber. Two studio loudspeakers (Mackie HR624) were set up in front of a distributed microphone array. Four directional microphones were arranged quadratically and they pointed towards the loudspeakers. Source signals were played back by a PCI sound card (SoundBlaster Live!). In all experiments, the amplified microphone signals were recorded at a sampling rate of 44.1 kHz. Subsequently, we converted the recordings to the sampling rate of our on-line BSS implementations (16 kHz). Two sets of audio data were processed

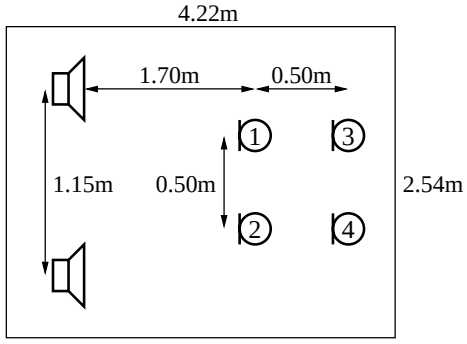

Fig. 4. Simulation setup (top view)

in our experiments:
Set 1 was created from a fairy tale. The narrator corresponds to source one and the actors to source two. Set 2 is based on an interview. Host and guests represent the source signals.

- **Experiment 1**: Fully synchronized data.
  In contrast to experiment 2 and 3, the microphone signals were recorded by one sound card (RME DIGI9652). One common internal clock controls the sampling rate, which is therefore identical for all microphones.

- **Experiment 2**: No synchronization.
  We recorded the microphone signals as well as the synchronization signal with the internal sound cards of four different laptops (Tab. 1). Synchronization signals were only used to find the common start position for all audio tracks.

- **Experiment 3**: Proposed synchronization model.
  In addition to experiment 2, we use the synchronization signal to align all the samples in the recorded signals.

Table 2 presents the average gain obtained due to BSS as a function of the choice of cost function, synchronization scheme, input audio signals, and the number of microphones. Experiment two clearly indicates that synchronization is an essential part in the (given) BSS methods. The performance of the BSS drops down significantly when compared to the

Mic	Laptop and sampling rate	
1	Dell Inspiron 7000	16001.7 Hz
2	IBM ThinkPad T20	16003.6 Hz
3	IBM ThinkPad 600E	16001.8 Hz
4	IBM ThinkPad T23	16009.5 Hz

Table 1. Recording devices and their corresponding sampling frequencies.

fully synchronized case. Our synchronization proposal addresses this issue.
We also found that for the case of two and three microphones and fully synchronized sampling, BSS based on cost function J_2 outperforms J_1-based BSS. J_2-based BSS falls lightly behind J_1-based BSS for four microphones. If data is not synchronized, J_1-based BSS show better performance. In experiment three, both BSS approaches yield similar gains.
An increased number of microphones improves BSS performance in the fully synchronized case. (Exception: Tab. 2d, three and four microphones) This additional gain vanishes if data is not synchronized. Using signals 'Interview', the gain may actually drop when more channels are used for the processing and the synchronization is not exact.

4. DISCUSSION AND CONCLUSION

Currently, array signal processing algorithms are restricted to single computing platform scenario. In this work we demonstrated how to distribute signal recording and processing to a network of general purpose computing devices, and examined the problem of sensor synchronization that arises in this case. Our synchronization method reduces the effects of different sampling rates. Further improvements of the proposed scheme are needed to obtain the best possible performance.

5. ACKNOWLEDGMENT

We would like to thank W. Herbordt and W. Kellermann from the Multimedia Communications and Signal Processing department (University Erlangen-Nürnberg) for valuable discussions while preparing this paper.

Experiment	Number of microphones 2	3	4
1: full sync	3.86 dB	5.29 dB	5.77 dB
2: no sync	1.58 dB	1.84 dB	1.84 dB
3: our sync	2.28 dB	2.69 dB	2.94 dB

(a) Fairy tale, Cost function J_1

Experiment	Number of microphones 2	3	4
1: full sync	4.88 dB	5.61 dB	5.76 dB
2: no sync	1.48 dB	1.61 dB	1.62 dB
3: our sync	2.33 dB	2.58 dB	2.75 dB

(b) Fairy tale, Cost function J_2

Experiment	Number of microphones 2	3	4
1: full sync	5.05 dB	6.90 dB	7.50 dB
2: no sync	2.75 dB	2.41 dB	2.35 dB
3: our sync	3.42 dB	3.25 dB	3.88 dB

(c) Interview, Cost function J_1

Experiment	Number of microphones 2	3	4
1: full sync	6.32 dB	7.33 dB	7.26 dB
2: no sync	1.93 dB	1.90 dB	1.88 dB
3: our sync	3.50 dB	3.45 dB	3.76 dB

(d) Interview, Cost function J_2

Table 2. Experimental results

6. REFERENCES

[1] M. Brandstein, D. Ward: *Microphone Arrays*, Springer Verlag, Berlin, May 2001.

[2] S. L. Gay, J. Benesty: *Acoustic Signal Processing for Telecommunication*, Kluwer Academic Publishers, March 2000.

[3] W. Herbordt, W. Kellermann: *Frequency-Domain Integration of Acoustic Echo Cancellation and a Generalized Sidelobe Canceller with Improved Robustness*, European Transactions on Telecommunications (ETT), vol. 13, no. 2, pages 123-132, March 2002.

[4] L. Parra, C. Spence: *On-line Blind Source Separation of Non-Stationary Signals*, Journal of VLSI Signal Processing, vol. 26, no. 1/2, pp. 39-46, August 2000.

[5] C. L. Fancourt, L. Parra: *The coherence function in blind source separation of convolutive mixtures of non-stationary signals*, IEEE International Workshop on Neural Networks and Signal Processing, 2001.

[6] D. D. Rife, J. Vanderkooy: *Transfer-Function Measurement with Maximum-Length Sequences*, J. Audio Eng. Soc., vol. 37, no. 6, June 1989.